An open source MATLAB program for fast numerical Feynman integral calculations for open quantum system dynamics on GPUs

Nikesh S. Dattani[1, *]

[1] *Physical and Theoretical Chemistry Laboratory, Department of Chemistry, University of Oxford, Oxford, OX1 3QZ, UK*

(Dated: $March 6, 2013$)

This MATLAB program calculates the dynamics of the reduced density matrix of an open quantum system modeled by the Feynman-Vernon model. The user gives the program a vector describing the coordinate of an open quantum system, a hamiltonian matrix describing its energy, and a spectral distribution function and temperature describing the environment's influence on it, in addition to the open quantum system's intial density matrix and a grid of times. With this, the program returns the reduced density matrix of the open quantum system at all (or some) moments specified by that grid of times. This overall calculation can be divided into two stages: the setup of the Feynman integral, and the actual calculation of the Feynman integral for time-propagation of the density matrix. When this program calculates this propagation on a multi-core CPU, it is this propagation that is usually the rate limiting step of the calculation, but when it is calculated on a GPU, the propagation is calculated so quickly that the setup of the Feynman integal actually becomes the rate limiting step for most cases tested so far. The overhead of transfrring information from the CPU to the GPU and back seems to have negligible effect on the overall runtime of the program. When the required information cannot fit on the GPU, the user can choose to run the entire program on a CPU.

INTRODUCTION

One is very often interested in how the reduced density matrix ρ of an **OQS** (**o**pen **q**uantum **s**ystem) changes with respect to time ($\rho D \equiv$reduced density matrix dynamics). The most popular open quantum system model to date is the Feynman-Vernon model (see section II), and for this model, a formally exact mathematical description of the ρD was deveoped by Richard P. Feynman and two of his PhD students Frank L. Vernon Jr. and Willard H. Wells in the late 1950s [4, 5, 22]. In this mathematical formalism, the ρD is represented in terms of Feynman integrals, for which no general closed form analytic solution is known. Therefore, to determine the numerical values of the elements of the reduced density matrix at a given point in time, one needs to evaluate these Feynman integrals numerically. Early work in numerically calculating these Feynman integrals used Monte Carlo methods, but since the integrnds are highly oscillatory, deterministic algorithms have been the most popular methods for these numercal Feynman integral calculations since breakthroughs in algorithmic development were made , mainly between 1992 and 2001.

The **qu**asi-**a**diabatic **p**ropagator Feynman[1] **i**ntegral (**QUAPI**) technique indroduced by Nancy Makri in 1992[9] helps the numerical calculation of these Feynman integrals converge with a larger sized time step used in the discretization of the Feynman integrals, when the bath is nearly adiabatic. Today's most popular algorithm for calculating these numerical Feynman integrals (whether using a quasi-adiabatic propagotor or not) uses a representation of the numerical Feynman integrals in terms of matrix-vector-like operations, and is called the 'tensor propagator scheme'. The most evolved form of the tensor propagator scheme was explained by Nancy Makri and Dmitrii E. Makarov in 1995 [12], although the original formulation of it was introduced by them in 1994[8] and explained in more detail in their 1995 paper[11]. The **f**iltered **p**ropagator **f**unctional (**FPF)** introduced by Eunji Sim and Nancy Makri in 1996[19**?**] is an improvement of the tensor propagator scheme which saves computer memory by using Monte Carlo importance sampling to filter which Feynman paths are included in the Feynman integral, but Eunji Sim's 2001 algorithm [18] is an alternative way of filtering the Feynman paths within the tensor propagator scheme which does not require Monte Carlo importance sampling and has been shown to save even more memory than the FPF technique, without sacrificing accuracy. I am not aware of any further algorithmic developments for the calculations of numerical Feynman integrals for the ρD of the Feynman-Vernon model in the first decade of the 21st century. Very recently another technique was introduced which can significantly reduce the number of Feynman paths required in the calculation of these Feynman integrals without significantly deteriorating the accuracy, which is particularly useful when the reduced density matrix changes so quickly that resolving its dynamics requires it to be calculated much more often than the response function of the OQS's environment needs to be sampled for a chosen amount of accuracy[1].

In the MATLAB program described in this paper, the main calculation which propagates the reduced density matrix in time using Feynman integration, is mainly carried out by the MATAB function BSXFUN (**b**inary **s**ingleton e**x**pansion **fun**ction), which was first introduced in March 2007 in version R2007a of MATLAB. This algorithm which uses BSXFUN is the fastest MATLAB implementation for the tensor propagator scheme metioned above, with which I have experimented. It is also extremely well suited for performance on **GPU**s (**g**raphical **p**rocessing **u**nits).

The company AccelerEyes developed the commercial software Jacket (for which the 1st beta version was v0.2 which was released in June 2008), which allows MATLAB functions to be performed with significant speedup on GPUs. BSXFUN was first incorporated into Jacket in v1.2, which was released in October 2009. In April 2010 a viral blog entry was published[**?**], entitled 'Crushing MATLAB Loop Runtimes with BSXFUN', which demonstrated that for the example calculation in that study, BSXFUN performed about 45 times faster than the naive MATLAB algorithm, when both calculations were executed on the same CPU. Even more impressively, when this BSXFUN calculaton was performed on their GPU, the calculation was sped up by almost another 6 times!

MATLAB itself did not have a built-in capability to execute caculations on GPUs until September 2010 in version R2010b, and it did not support the use of the built-in BSXFUN function on GPUs until very recently (March 2012 in version R2012a).

SETTING

The most popular OQS model is currently the Feynman-Vernon model. In the Feynman-Vernon model, the OQS (whose coordinate is denoted by s) is coupled linearly to a set of **QHO**s (**q**uantum **h**armonic **o**scillators) Q_k:

$$H = H_{\mathrm{OQS}} + H_{\mathrm{OQS-bath}} + H_{\mathrm{bath}} \tag{1}$$

$$= H_{\mathrm{OQS}} + \sum_{\kappa} c_\kappa s Q + \sum_{\kappa} \left(\tfrac{1}{2} m_\kappa \dot{Q_\kappa}^2 + \tfrac{1}{2} m_\kappa \omega_\kappa^2 Q_\kappa^2 \right). \tag{2}$$

In most models the QHOs span a continuous spectrum of frequencies ω_κ and the strength of the coupling between the QHO of frequency ω and the OQS is given by the spectral distribution function $J(\omega)$:

$$J(\omega) = \frac{\pi}{2} \sum_{\kappa} \frac{c_\kappa^2}{m_\kappa \omega_\kappa} \delta(\omega - \omega_\kappa). \tag{3}$$

[1] The term 'path integral' is used more commonly than 'Feynamn integral' here, but this term is ambiguous. Currently, the first result on the search engine at www.google.com, when the search query 'path integral' is entered, is a Wikipedia page that currently links to three different meanings of the word 'path integral': (1) line integral, (2) functional integration, and (3) path integral formulation. Only the third of these is unambiguously the Feynman integral discussed in this paper. The 'line integral' is an integral over a path, rather than over a set of paths; and the term 'functional integral' can refer to at least three types of functional integrals: (1) the Wiener integral, (2) the Lévy integral, and (3) the Feynman integral.

For the hamiltonian of the Feynman-Vernon model, the bath response function $\alpha(t)$ is the following integral transform of $J(\omega)$, in terms of the inverse temperature $\beta \equiv \frac{1}{k_B T}$:

$$\alpha(t) = \frac{1}{\pi}\int_0^\infty J(\omega)\left(\coth\left(\frac{\beta\omega\hbar}{2}\right)\cos(\omega t) - \mathrm{i}\sin(\omega t)\right)\mathrm{d}\omega \tag{4}$$

$$= \frac{1}{\pi}\int_{-\infty}^\infty \frac{J(\omega)\exp\left(\frac{\beta\omega\hbar}{2}\right)}{2\sinh\left(\frac{\beta\omega\hbar}{2}\right)} e^{-\mathrm{i}\omega t}\mathrm{d}\omega,\ J(-\omega)\equiv J(\omega) \tag{5}$$

$$= \frac{1}{\pi}\int_{-\infty}^\infty \frac{J(\omega)}{1-\exp(-\beta\omega\hbar)} e^{-\mathrm{i}\omega t}\mathrm{d}\omega\,,\ J(-\omega)\equiv J(\omega), \tag{6}$$

where, equation 6 can be written in terms of the Bose-Einstein distribution function with $x = \beta\omega\hbar$:

$$f^{\text{Bose–Einstein}}(x) = \frac{1}{1-\exp(-x)}. \tag{7}$$

Assuming that the density matrix of the OQS plus its environment at $t = 0$ is $\rho_{\text{total}} = \rho(0)\otimes e^{-\beta H_{\text{bath}}}$, the reduced density matrix elements at time $t = N\cdot\Delta t$ are given exactly by[10] (the integrals are used if the coordinate s of the OQS is continuous, and the summations are used if it is discrete or discretized):

$$\langle s_N^+|\rho(t)|s_N^-\rangle = \oint\!\!\!\!\!\!\int \lim_{\substack{\Delta t\to 0\\ \Delta k_{\max}\to\infty}} \left(\prod_{k=0}^{N-1}\langle s_{k+1}^+|e^{-\frac{\mathrm{i}}{\hbar}H_{\text{OQS}}\Delta t}|s_k^+\rangle\langle s_k^-|e^{\frac{\mathrm{i}}{\hbar}H_{\text{OQS}}\Delta t}|s_{k+1}^-\rangle\right)\langle s_0^+|\rho(0)|s_0^-\rangle I\left(\{s_k^\pm\}_{k=0}^N;\Delta t\right)\prod_{k=0}^{N-1}\mathrm{d}s_k^+\mathrm{d}s_k^-\ , \tag{8}$$

where the discretized influence functional[2] can be written as:

$$I = \exp\left(-\sum_{k=0}^{\Delta k_{\max}}\sum_{k'=0}^{k}(s_k^+ - s_k^-)(\eta_{kk'}s_{k'}^+ - \eta_{kk'}^* s_{k'}^-)\right)\ , \tag{9}$$

and the η coefficients are given in terms of the bath response function $\alpha(t)$ by [2]:

$$\eta_{kk'} \equiv \int_{k\Delta t}^{(k+1)\Delta t}\int_{k'\Delta t}^{(k'+1)\Delta t}\alpha(t'-t'')\mathrm{d}t''\mathrm{d}t'\ ,\ k\neq k'\ , \tag{10}$$

$$\eta_{kk} \equiv \int_{k\Delta t}^{(k+1)\Delta t}\int_{k\Delta t}^{t'}\alpha(t'-t'')\mathrm{d}t''\mathrm{d}t'\ ,\ k\notin\{0,N\}\ , \tag{11}$$

$$\eta_{N0} \equiv \int_{N\Delta t-\Delta t/2}^{N\Delta t}\int_0^{\Delta t/2}\alpha(t'-t'')\mathrm{d}t''\mathrm{d}t'\ , \tag{12}$$

$$\eta_{00} \equiv \int_0^t\int_0^{t'}\alpha(t'-t'')\mathrm{d}t''\mathrm{d}t'\ , \tag{13}$$

$$\eta_{NN} \equiv \int_{N\Delta t-\Delta t/2}^{N\Delta t}\int_{N\Delta t-\Delta t/2}^{t'}\alpha(t'-t'')\mathrm{d}t''\mathrm{d}t'\ , \tag{14}$$

$$\eta_{k0} \equiv \int_{k\Delta t}^{(k+1)\Delta t}\int_0^{\Delta t/2}\alpha(t'-t'')\mathrm{d}t''\mathrm{d}t'\ , \tag{15}$$

$$\eta_{Nk} \equiv \int_{N\Delta t-\Delta t/2}^{N\Delta t}\int_{k\Delta t}^{(k+1)\Delta t}\alpha(t'-t'')\mathrm{d}t''\mathrm{d}t'\ , \tag{16}$$

and can very often be represented by closed form analytic expressions[2, 3]. The purpose of the program is to calculate all (or some) of the matrix elements $\langle s_N^+|\rho(t)|s_N^-\rangle$ at a time t (or at all of the times $\{t_k = k\cdot\Delta t\}_{k=0}^N$), given the system coordinate s, its hamiltonian H_{OQS} and initial density matrix $\rho(0)$, the spectral distrution function $J(\omega)$ **or** the bath response function $\alpha(t)$, the temperature T, and the convergence parameters $\{\Delta t, \Delta k_{\max}\}$.

To save time in recalucalting certain quantities involved in the overall calculation, the program stores four variables when $N > \Delta k_{\max}$, each containing

$$M^{2(\Delta k_{\max}+1)} \tag{17}$$

[2] which is actually a function

(double precision) floating point complex numbers (here M is the dimension of the Hilbert space of the OQS). In my experience, using single precision numbers has sometimes given values for $\rho(t)$ that are very slightly (but observably) different from when double precision numbers are used, and since the purpose of the numerical Feynman integral technique described here is to calculate *exact* values of $\rho(t)$ (usually for benchmarking less accurate, less computationally expensive methods), double precision is recommended (although the program will still work if one desires to keep less digits to save memory). When these complex numbers are double precision, they take up 16 bytes of the computer's memory, so the overall amount of memory required to store the four variables mentioned just before equation 17 is

$$\begin{aligned} \mathrm{PMC} &= 4 \cdot 16 \cdot M^{2(\Delta k_{\max}+1)} && (18) \\ &= 64 \cdot M^{2(\Delta k_{\max}+1)} && (19) \end{aligned}$$

bytes of information overall (here **PMC** stands for **p**rimay **m**emory **c**ost, since the storage of these numbers almost always constitutes the vast majority of the memory required for the overall calculation, and the PMC is the most important computational complexity measure associated with the program described in this paper). This memory cost could be reduced if one of the filtering techniques mentioned in the introduction section of this paper were to be implemented; however the current version of the program does not have either of these algorithms implemented.

MODEL SYSTEM FOR EXAMPLE CALCULATIONS

All example calculations in this paper will be for a model system with:

$$\begin{aligned} s &\in \{0,1\} \ , && (20) \\ H_{\mathrm{OQS}} &= \frac{1}{2}\begin{pmatrix} 0 & \Omega(t) \\ \Omega(t) & 0 \end{pmatrix} \ , \text{and} && (21) \\ J(\omega) &= A\omega^3 e^{(-\omega/\omega_c)^2} \ . && (22) \end{aligned}$$

This OQS model has been used to describe many experiments (some examples are [16]) on laser-driven single quantum dots after a **r**otating **w**ave **a**pproximation (**RWA**). The specific parameters used in this paper will consistently be:

$$\begin{aligned} \Omega(t) &= \pi/8 \ , \\ A &= \pi 0.027\mathrm{ps}^2 \ , && (23) \\ \omega_c &= 2.2\mathrm{ps}^{-1} \ , \text{and} && (24) \\ T &= 25\mathrm{K} \ . \end{aligned}$$

The values of A and ω_c come from the experimental study of this model system described in [16], and have been used in many computational studies that use this same model system (some examples are [1, 13, 14]). The values of Ω and T were chosen in order for the damped Rabi oscillations to last for a long time (for better evaluation of the performance of the program's propagation of ρ with respect to time), while being damped enough to be physically interesting, and while keeping Ω and T physically realizable.

PERFORMANCE

All calculations reported in this paper were performed on an Intel Xeon/Nehalem CPU with 2.8GHz clock rate, and/or on an Nvidia C2020 GPU which are part of the SKYNET Viglen/NVIDIA hybrid GPGPU Cluster provided by the Oxford Supercomputing Centre at the Oxford e-Research Centre at University of Oxford. This this cluster is continually in use by many people, making its performance variant with respect to time. This may be reflected in some of the performance times reported below, but it is still valuable to see the overall trends and comparisons.

Table I. Running the main propagation calculations ($k > \Delta k$)) on the CPU vs the GPU for varying values of $\Delta k_{\max}$. The pimary memory cost (PMC) associated with each value of $\Delta k_{\max}$ is given by equation 19.

$\Delta k_{\max}$	PMC	Run time on CPU (seconds)	Run time on GPU (seconds)	Approximate speedup on GPU
2	4.096 KB	0.071580	1.224592	
3	16.38 KB	0.081609	1.226297	
4	65.54 KB	0.107413	1.230001	
5	262.1 KB	0.209927	1.22675	
6	1.049 MB	0.641943	2.177962	
7	4.194 MB	2.571893	1.286302	2x
8	16.78 MB	16.347692	1.766242	9x
9	67.11 MB	59.452344	4.673941	13x
10	268.4 MB	291.731064	15.580798	19x
11	1.074 GB	1198.620937	21.130031	18x
12	4.295 GB	4911.546824	Required data did not fit on the GPU	

Based on the results in table I, we see that the speed benefit gained by running the main propagation calculations on the GPU rather than the CPU become more and more apparent as $\Delta k_{\max}$ is increased, until $\Delta k_{\max} = 11$. For this reason, the performance tests reported below are for calculations where $\Delta k_{\max} = 11$.

Figure 1 presents an estimate of the amount of time spent by the program on each line of the part of the code surrounding the lines that execute the main propagation calculations (lines 170-171 for the CPU version of the program, and lines 164-165 for the GPU version). The calculation of the summation to determine ρ at the times $\{t_k = k \cdot \Delta t\}_{k=\Delta k_{\max}}^{N}$ is executed on line 180 for the CPU version and 169 for the GPU version.

Figure 1. Top: Profiler output when the main propagation calculations are performed on the CPU (top) and GPU (bottom).

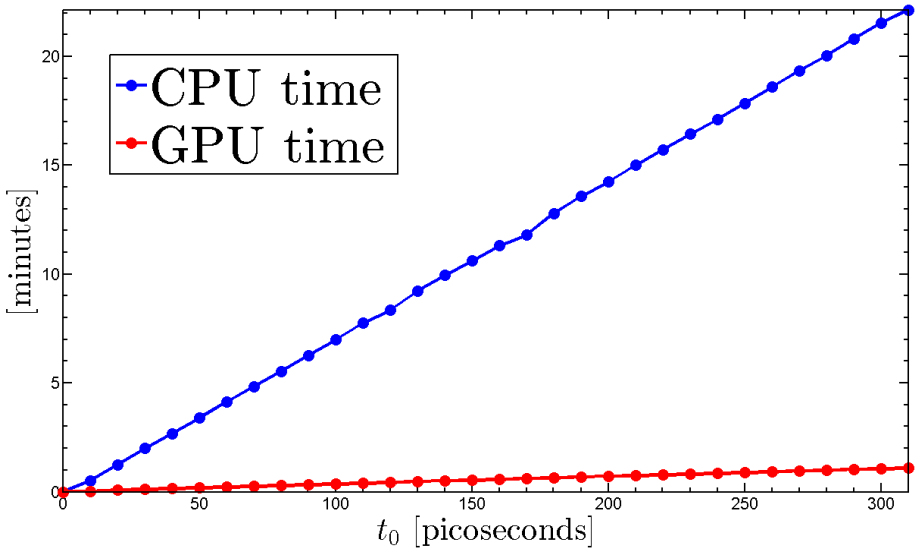
```
 11.95      12  156      KtimesIpermanent=KtimesIpermanent.*I_mk((indices(:,end-k)-1)*M2+indices(:,end),k+1); %whe
 11.74      12  157      KtimesIpermanentEnd=KtimesIpermanentEnd.*I_mkEnd((indices(:,end-k)-1)*M2+indices(:,end),k
            12  158  end
                159
< 0.01       1  160  indices=[]; %clear('indices') %when not using parfor
             1  167  KtimesIpermanent=reshape(KtimesIpermanent,M2,[]);
             1  168  KtimesIpermanentEnd=reshape(KtimesIpermanentEnd,M2,[]);
             1  169  tic;
             1  170  for J=deltaKmax+1:finalPoint
523.75     989  171  Aend=reshape(bsxfun(@times,reshape(sum(reshape(A,[],M2),2),1,[]),KtimesIpermanentEnd),[],1);
524.25     989  172  A=reshape(bsxfun(@times,reshape(sum(reshape(A,[],M2),2),1,[]),KtimesIpermanent),[],1); %coul
  0.02     989  179  Aend=reshape(Aend,M2,[]);
130.06     989  180  rho(1,J+1)=sum(Aend(1,:),2);
```

```
 12.68      12  151      KtimesIpermanent=KtimesIpermanent.*I_mk((indices(:,end-k)-1)*M2+indices(:,end),k+1); %when k
 12.38      12  152      KtimesIpermanentEnd=KtimesIpermanentEnd.*I_mkEnd((indices(:,end-k)-1)*M2+indices(:,end),k+1)
            12  153  end
                154
< 0.01       1  155  indices=[]; %clear('indices') %when not using parfor
                156
             1  157  KtimesIpermanent=reshape(KtimesIpermanent,M2,[]);
             1  158  KtimesIpermanentEnd=reshape(KtimesIpermanentEnd,M2,[]);
             1  159  tic;
  0.10       1  160  KtimesIpermanent=gpuArray(KtimesIpermanent);
  0.10       1  161  KtimesIpermanentEnd=gpuArray(KtimesIpermanentEnd);
  0.10       1  162  A=gpuArray(A);
< 0.01       1  163  rho=gpuArray(rho(1,:));
             1  164  for J=deltaKmax+1:finalPcint
  0.49     989  165  Aend=reshape(bsxfun(@times,reshape(sum(reshape(A,[],M2),2),1,[]),KtimesIpermanentEnd),[],1); %se
  0.17     989  166  A=reshape(bsxfun(@tines,reshape(sum(reshape(A,[],M2),2),1,[]),KtimesIpermanent),[],1); %could be
                167
  0.02     989  168  Aend=reshape(Aend,M2,[]);
 42.30     989  169  rho(1,J-1)=sum(Aend(1,:),2);
```

The most striking observation from figure 1 is that while on the CPU version, the propagation calculations (lines 170-171) are the obvious bottleneck, when these calculations are done on the GPU (lines 165-166 of the GPU version), the profiler estimates that these calculations take less than a second in total (despite being run 989 times each) ! This means that the new bottleneck for the GPU version is the summation on line 169, that caculates ρ at 989 successive points in time, and according to the profiler's estimate takes 42.3 seconds in total (0.0428 seconds for each time value at which ρ is calculated).

Very often, one is only interested in $\rho(t)$ after a certain amount of time t_{specific} has passed. For example, in the study of laser-driven quntum dots, one is often interested in the population of excitons after a certain amount of time (given by $\langle 1|\rho(t_{\text{specific}})|1\rangle = \rho_{11,t_{\text{specific}}}$

Figure 2. CPU time and GPU time vs t_{specific} when ρ is only calculated at the one time $t_0 \equiv t_{\text{specific}}$.



in the model studied in this paper), and how this quantity changes with respect to a physical parameter. The experimental data in figure 1 of [17] and in figures 1 and 2 of [16] report the photocurrent at a specific time t_{specific} (the photocurrent is related to $\rho_{11,t_{\text{specific}}}$), as a function of the 'pulse area' (which is related to $\Omega(t)$ in our hamiltonian given by equation 21) of the laser that drives a quantum dot. Various figures in theoretical and computatonal studies of this same system (see for example [6, 7, 13, 14, 20, 21], and many more) present $\rho_{11,t_{\text{specific}}}$ as a function of the pulse area. In all of these cases, one is not interested in $\rho(t)$ at every point in time, but only in $\rho(t_{\text{specific}})$, so line 169 of the GPU version (or line 180 of the CPU vesion) of the code in figure 1 would only have to be executed once instead of 989 times, for each pulse area for which one desires to know $\rho(t_{\text{specific}})$.

For this reason, the program described in this paper allows the user to specify the input variable `allPointsOrJustFinalPoint` by either the string `'allPoints'`, which runs line 180 of the CPU version, or line 169 of the GPU version, for all values of $\{t = k{\cdot}\Delta t\}_{k=\Delta k_{\max}}^{N}$; or by the string `'justFinalPoint'`, which runs those lines only once, at $k = N = \frac{t_{\text{specific}}}{\Delta t}$. All calculations for which the performance was reported in figure 2 were run with `allPointsOrJustFinalPoint = 'justFinalPoint'`. Figure 2 not only demonstrates that the benefit of running the main propagation steps on the GPU rather than the CPU grow with the amount of time for which the system is being simulated, but also demonstrates very clearly the linear scaling of the markovian tensor propagator algorithm of Makri and Makarov.

DISCUSSION

It is worth noting that in the program described in this paper, based on the profiler output in the bottom panel of figure 1, the main propagation calcualtions and each summation used to determine the numerical values of ρ at a particular time t, take a very small (almost negligible) amount of time to compute. This means that other parts of the program, which set up the numerical Feynman integral calculation (such as the calculation of the η coefficients presented in equations 10 to 16) can play an equally important, or dominant role to the overall runtime of the program (especially if not implemented efficiently). In figure 1 of [2], it was shown that the calculation of $\eta_{kk'}$, $k \neq k'$ (equation 10 of this paper), for the same spectral distribution function used in the calculations presented in this paper (given by equations 22, 23 and 24), took more than 300 seconds to calculate on the CPU used when using Mathematica 8's `NIntegrate` function to numerically integrate the most frequently used expressions for these quantities, while it took less than half a second to compute on the same CPU using Mathematica 8's `Sum` function to evaluate the expression introduced in [2] for the same quantities. For this reason, using the analytic expressions for the η coefficients first given in [2] will help to maximize the benefit from the GPU version of the program described in this paper - and if one were to numerically compute the historically more traditional expressions for these coefficients, the time taken to execute this numerical integration would in many cases dominate the execution time for the rest of the program's calculations.

It is also important to discuss the computational overhead associated with transfering data from the CPU to the GPU and back, especially the GPU deals with a very large[3] amount of data for the calculations performed by the program described in this paper. The transferring of the relevant data from the CPU to the GPU is done by lines 160-163 of the GPU version of the code, and the transferring of the result $\{\rho(t_k)\}$ stored in the array `rho`, is transferred back from the GPU to the CPU on line 171 of that code. The result of the profiler's estimates if runtime displayed in the bottom panel of figure 1 suggest that the transferring of the data to and from the GPU is performed in real time. The profiler is perhaps underestimating the runtimes of at least one of the lines of code

[3] When the accuracy of the calculations is important, the size of the data involved in the PMC will ideally be as large as the GPU's memory will allow.

performed on the GPU (between lines 160 and 172). This can be seen by observing that figure 2 shows that lines 160 to 172 of the GPU version of the code can take up to one minute, and the sum of all the runtimes estimated by the profiler for these lines never reached one minute when these calculations were redone with the profiler on[4]. If it is true that the profiler has underestimated the runtimes of lines 160-163 in figure 1, which transfer the relevant data from the CPU to the GPU, this might at least partially explain why line 165 was reported to have taken significantly longer than line 166 in the bottom panel of figure 1, despite the top panel of the same figure showing that in the analogous lines for the CPU version of the code (lines 171 and 172), line 172 is actually faster than 171 ! In any case though, the transfer of the data to and from the GPU cannot be significant to the overall runtime of the program, since the exact same data is transfered in every instance represented by a red circle in figure 2, and therefore the overall runtime associated with transferring the data should be within the scale of the first red circle. The linear increase seen in the red curve is due to the fact that lines 165 and 166 are repeated more and more times in the `for` loop as N is increased.

The program described in this paper is open source, and users are encouraged to contribute on Github by implementing techniques to improve its efficiency, such as the OFPF technique [18] and the interpolation method [1] mentioned in the introduction, and the extension to allow for models where the OQS can couple to more than one QHO of a given frequency using the more general influence functional first presented in [15].

ACKNOWLEDGEMENTS

I would like to express my most warm thanks to Professor Mike Giles, Dr. Wes Armour, Dr. Andrew Richards, and the rest of the **O**xford **S**upercomputing **C**entre (**OSC**) and **O**xford **e**-**R**esearch **C**entre (**OeRC**) for allowing me to use their Viglen/NVIDIA hybrid GPGPU Cluster SKYNET. I am also delighted to thank Dr. Mihai Duta, Albert Solernou, Steven Young, and the rest of the Oxford e-Research Centre staff for helping me get started with using the GPUs on SKYNET, and for being such friendly and helpful support staff for the OSC and OeRC supercomputers. I also thank the many members of the MATLAB Central Newsgroup who helped me develop the MATLAB program described in this paper, particularly Matt J and James Tursa for introducing me to BSXFUN; and I thank Gallagher Pryor for his blog entry which immediately brought the power of implementing BSXFUN on GPUs to my very keen attention. Finally, I thank the Clarendon Fund and the NSERC/CRSNG of/du Canada for financial support.

* nike.dattani@chem.ox.ac.uk

[1] Nikesh S. Dattani. Numerical Feynman integrals with physically inspired interpolation: Faster convergence and significant reduction of computational cost. *AIP Advances*, 2(1):012121, 2012.
[2] Nikesh S Dattani, Felix A Pollock, and David M Wilkins. Analytic Influence Functionals for Numerical Feynman Integrals in Most Open Quantum Systems. *Quantum Physics Letters*, 1(1):35–45, 2012.
[3] Nikesh S. Dattani, David M. Wilkins, and Felix A. Pollock. Optimal representation of the bath response function & fast calculation of influence functional coefficients in open quantum systems with BATHFIT 1. page 6, May 2012.
[4] Richard Phillips Feynman, Albert R Hibbs, and Daniel F Styer. *Quantum Mechanics and Path Integrals*. Dover Publications Inc., emended edition, 2010.
[5] Richard Phillips Feynman and F L Vernon Jr. The Theory of a General Quantum System Interacting with a Linear Dissipative System. *Annals of Physics*, 24:118–173, 1963.
[6] M. Glässl, M. Croitoru, A. Vagov, V. Axt, and T. Kuhn. Influence of the pulse shape and the dot size on the decay and reappearance of Rabi rotations in laser driven quantum dots. *Physical Review B*, 84(12), September 2011.
[7] S. Lüker, K. Gawarecki, D. Reiter, A. Grodecka-Grad, V. Axt, P. Machnikowski, and T. Kuhn. Influence of acoustic phonons on the optical control of quantum dots driven by adiabatic rapid passage. *Physical Review B*, 85(12), March 2012.
[8] Dmitrii E Makarov and Nancy Makri. Path integrals for dissipative systems by tensor multiplication. Condensed phase quantum dynamics for arbitrarily long time. *Chemical Physics Letters*, 221:482–491, 1994.
[9] N MAKRI. IMPROVED FEYNMAN PROPAGATORS ON A GRID AND NONADIABATIC CORRECTIONS WITHIN THE PATH INTEGRAL FRAMEWORK. *CHEMICAL PHYSICS LETTERS*, 193(5):435–445, June 1992.
[10] N MAKRI. NUMERICAL PATH-INTEGRAL TECHNIQUES FOR LONG-TIME DYNAMICS OF QUANTUM DISSIPATIVE SYSTEMS. *JOURNAL OF MATHEMATICAL PHYSICS*, 36(5):2430–2457, May 1995.
[11] Nancy Makri and Dmitrii E Makarov. Tensor propagator for iterative quantum time evolution of reduced density matrices. I. Theory. *Journal of Chemical Physics*, 102(11):4600–4610, 1995.
[12] Nancy Makri and Dmitrii E Makarov. Tensor propagator for iterative quantum time evolution of reduced density matrices. II. Numerical methodology. *Journal of Chemical Physics*, 102(11):4611–4618, 1995.
[13] Dara P S McCutcheon and Ahsan Nazir. Quantum dot Rabi rotations beyond the weak exciton-phonon coupling regime. *New Journal of Physics*, 12(11):113042, November 2010.
[14] DPS McCutcheon, NS Dattani, EM Gauger, BW Lovett, and A Nazir. A general approach to quantum dynamics using a variational master equation: Application to phonon-damped Rabi rotations in quantum dots. *Physical Review B*, 84(1):2–5, 2011.
[15] P Nalbach, J Eckel, and M Thorwart. Quantum coherent biomolecular energy transfer with spatially correlated fluctuations. *New Journal of Physics*, 12(6):065043, June 2010.

[4] Except for the calculations for which the profiler results were displayed in figure 1, all calculations reported in this paper were performed with without the profiler running, in order to remove the confusion that would be introduced due to the computational overhead of using the profiler (although this overhead was in fact found to be very small).

[16] A. Ramsay, T. Godden, S. Boyle, E. Gauger, A. Nazir, B. Lovett, A. Fox, and M. Skolnick. Phonon-Induced Rabi-Frequency Renormalization of Optically Driven Single InGaAs/GaAs Quantum Dots. *Physical Review Letters*, 105(17), October 2010.

[17] A J Ramsay, Achanta Venu Gopal, E M Gauger, A Nazir, B W Lovett, A M Fox, and M S Skolnick. Damping of Exciton Rabi Rotations by Acoustic Phonons in Optically Excited In GaAs/GaAs Quantum Dots. *Phys. Rev. Lett.*, 104(1):17402, January 2010.

[18] Eunji Sim. Quantum dynamics for a system coupled to slow baths: On-the-fly filtered propagator method. *Journal of Chemical Physics*, 115(10):4450–4456, 2001.

[19] Eunji Sim and Nancy Makri. Tensor propagator with weight-selected paths for quantum dissipative dynamics with long-memory kernels. *Chemical Physics Letters*, 249:224–230, 1996.

[20] A Vagov, M D Croitoru, Vollrath Martin Axt, T Kuhn, and F M Peeters. Nonmonotonic field dependence of damping and reappearance of Rabi oscillations in quantum dots. *Physical Review Letters*, 98(22):227403(1–4), 2007.

[21] A Vagov, M D Croitoru, M Glässl, V M Axt, and T Kuhn. Real-time path integrals for quantum dots: Quantum dissipative dynamics with superohmic environment coupling. *Phys. Rev. B*, 83(9):94303, March 2011.

[22] Willard H Wells. Quantum Formalism Adapted to Radiation in a Coherent Field. *Annals of Physics*, 12:1–40, 1961.